IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Docket No. AUS9-2000-0587-US1

Assistant Commissioner for Patents
Washington, D.C. 20231

Sir:

Transmitted herewith for filing is the patent application of Inventor(s):

Geoffrey Owen Blandy

For: **APPARATUS AND METHOD FOR CREATING INSTRUCTION BUNDLES IN AN EXPLICITLY PARALLEL ARCHITECTURE**

Enclosed are also:

X   ℨℨ   Pages of Specification including an Abstract
X   9   Pages of Claims
X   6   Sheet(s) of Drawings
X   A Declaration and Power of Attorney
X   Form PTO 1595 and assignment of the invention to IBM Corporation

CLAIMS AS FILED

FOR	Number Filed		Number Extra		Rate		Basic Fee ($690)
Total Claims	39	-20 =	19	X	$ 18	=	$342.00
Independent Claims	3	-3 =	0	X	$ 78	=	$0.00
Multiple Dependent Claims	0			X	$260	=	$
				Total Filing Fee		=	$1,032.00

X   Please charge $1,032.00 to IBM Corporation, Deposit Account No. 09-0447.

X   The Commissioner is hereby authorized to charge payment of the following fees associated with the communication or credit any over payment to IBM Corporation, Deposit Account No. 09-0447. A duplicate copy of this sheet is enclosed.

   X   Any additional filing fees required under 37CFR § 1.16.
   X   Any patent application processing fees under 37CFR § 1.17.

Respectfully,

Voel Emile
Reg. No. 39,969
Intellectual Property Law Dept.
IBM Corporation
11400 Burnet Road 4054
Austin, Texas 75758
Telephone: (512) 823-1005

Docket No. AUS9-2000-0587-US1

APPARATUS AND METHOD FOR CREATING INSTRUCTION BUNDLES IN AN EXPLICITLY PARALLEL ARCHITECTURE

RELATED APPLICATIONS

5

The present invention is related to commonly assigned and co-pending U.S. Patent Applications _____ (Attorney Docket No. AUS9-2000-0569) entitled "APPARATUS AND METHODS FOR IMPROVED DEVIRTUALIZATION OF METHOD

10 CALLS", _____ (Attorney Docket No. AUS9-2000-0570) entitled "APPARATUS AND METHOD FOR AVOIDING DEADLOCKS IN A MULTITHREADED ENVIRONMENT", _____ (Attorney Docket No. AUS9-2000-0572) entitled "APPARATUS AND METHOD FOR IMPLEMENTING SWITCH INSTRUCTIONS IN AN IA64

15 ARCHITECTURE", _____ (Attorney Docket No. AUS9-2000-0573) entitled "APPARATUS AND METHOD FOR DETECTING AND HANDLING EXCEPTIONS", _____ (Attorney Docket No. AUS9-2000-0584) entitled "APPARATUS AND METHOD FOR VIRTUAL REGISTER MANAGEMENT USING PARTIAL DATA FLOW

20 ANALYSIS FOR JUST-IN-TIME COMPILATION", _____ (Attorney Docket No. AUS9-2000-0585) entitled "APPARATUS AND METHOD FOR AN ENHANCED INTEGER DIVIDE IN AN IA64 ARCHITECTURE", and _____ (Attorney Docket No. AUS9-2000-0586) entitled "APPARATUS AND METHOD FOR CREATING INSTRUCTION GROUPS FOR

25 EXPLICITLY PARALLEL ARCHITECTURES", filed on even date herewith and hereby incorporated by reference.

BACKGROUND OF THE INVENTION

30 **1.   Technical Field:**

The present invention is directed to an apparatus

and method for creating instruction bundles in an explicitly parallel architectures. More particularly, the present invention is directed to an apparatus and method for creating instruction bundles for the IA64

5   architecture.

2.   Description of Related Art:

Explicitly parallel architectures, such as IA64, require that instructions be organized into bundles

10 comprising three instruction slots and a template field that identifies for each slot the execution unit type to which the instruction will be dispatched. Only a subset of instruction combinations are valid.

Because dynamic compilers, such as Just-In-Time

15 compilers, typically compile methods as they are invoked, compile time is a direct contributor to response time and therefore, should be minimized. At the same time, the type of bundles selected have a direct effect on the execution time of the compiled method.

20   Thus, it would be beneficial to have an apparatus and method for quickly creating instruction bundles that will maximize instruction level parallelism and thereby optimize performance of the compiled method.

Docket No. AUS9-2000-0587-US1

SUMMARY OF THE INVENTION

5   The present invention provides an apparatus and
method for creating instruction groups for explicitly
parallel architectures, and in particular,
implementations of the IA64 architecture. The apparatus
and method accept instruction groups as input. For each
10   instruction group, bundles are created based on the types
of instructions present in the group. The final bundle
of each group will contain a stop bit to designate the
end of the instruction group. In some cases the bundling
process of one group is not completed until a subsequent
15   group is examined to see if some or all of its
instructions may be placed in the final bundle of the
first group. When groups are combined in this way an
intra-bundle stop bit is used to designate the end of the
first instruction group. The instruction bundling is
20   performed based on a most restrictive instruction type
placement first and proceeds to less restrictive
instruction type placement.

Docket No. AUS9-2000-0587-US1

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The

5 invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

10 **Figure 1** is an exemplary block diagram of a data processing system according to the present invention;

Figure 2 is an exemplary diagram illustrating template field encoding and instruction slot mapping in accordance with an IA64 architecture;

15 **Figures 3A-3C** are diagrams illustrating pseudo-code for creating instruction groups for an explicitly parallel architecture in accordance with the present invention; and

Figure 4 is a flowchart outlining an exemplary

20 operation of the present invention.

Docket No. AUS9-2000-0587-US1

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

With reference now to the figures, and in particular
Figure 1, a block diagram of a data processing system in
5   which the present invention may be implemented is
illustrated. Data processing system **250** is an example of
a client computer, however, the present invention may be
implemented in a server, stand-alone computing device, or
the like. In short, the present invention may be
10   implemented in any data processing device having an
explicitly parallel architecture. By explicitly parallel
architecture, what is meant is that the compiler or
programmer is responsible for designating which
instructions may be executed in parallel. The
15   architecture provides a means for the compiler to
identifying such groups of instructions. For example, in
the IA64 architecture, described in greater detail
hereafter, the stop bits provide this means for
identifying groups of instructions.
20   Data processing system **150** employs a peripheral
component interconnect (PCI) local bus architecture.
Although the depicted example employs a PCI bus, other
bus architectures such as Micro Channel and ISA may be
used. Processor **152** and main memory **154** are connected to
25   PCI local bus **156** through PCI Bridge **158**. PCI Bridge **158**
also may include an integrated memory controller and
cache memory for processor **152**. Additional connections
to PCI local bus **156** may be made through direct component
interconnection or through add-in boards. In the
30   depicted example, local area network (LAN) adapter **160**,
SCSI host bus adapter **162**, and expansion bus interface

Docket No. AUS9-2000-0587-US1

164 are connected to PCI local bus **156** by direct
component connection. In contrast, audio adapter **166**,
graphics adapter **168**, and audio/video adapter (A/V) **169**
are connected to PCI local bus **166** by add-in boards

5   inserted into expansion slots. Expansion bus interface
164 provides a connection for a keyboard and mouse
adapter **170**, modem **172**, and additional memory **174**. SCSI
host bus adapter **162** provides a connection for hard disk
drive **176**, tape drive **178**, and CD-ROM **180** in the depicted

10   example. Typical PCI local bus implementations will
support three or four PCI expansion slots or add-in
connectors.

An operating system runs on processor **152** and is
used to coordinate and provide control of various

15   components within data processing system **150** in **Figure 1**.
The operating system may be a commercially available
operating system such as OS/2, which is available from
International Business Machines Corporation.

An object oriented programming system such as Java

20   may run in conjunction with the operating system and may
provide calls to the operating system from Java programs
or applications executing on data processing system **150**.
Instructions for the operating system, the object
oriented operating system, and applications or programs

25   are located on storage devices, such as hard disk drive
176 and may be loaded into main memory **154** for execution
by processor **152.** Hard disk drives are often absent and
memory is constrained when data processing system **150** is
used as a network client.

30   Those of ordinary skill in the art will appreciate
that the hardware in **Figure 1** may vary depending on the

Docket No. AUS9-2000-0587-US1

implementation. For example, other peripheral devices, such as optical disk drives and the like may be used in addition to or in place of the hardware depicted in **Figure 1**. The depicted example is not meant to imply

5 architectural limitations with respect to the present invention. For example, the processes of the present invention may be applied to a multiprocessor data processing system.

The present invention provides an apparatus and

10 method for creating instruction bundles for explicitly parallel architectures. In particular, the present invention provides an apparatus and method for creating instruction bundles for implementations of the IA64 explicitly parallel architecture. The IA64 architecture

15 is described in the "Intel IA-64 Architecture Software Developer's Manual" available for download from http://developer.intel.com/design/Ia-64/downloads /24531702s.htm, which is hereby incorporated by reference. While the present invention will be described

20 with reference to the Itanium implementation of the IA64 architecture, the present invention is not limited to such. Rather, the present invention is applicable to any explicitly parallel architecture and any implementation of the IA64 architecture in particular.

25 An IA64 program consists of a sequence of instructions and stops packed in bundles. A bundle is 128 bits in size and contains 3 41-bit instruction slots and a 5 bit template. The template maps the instruction slots to the execution units to which they will be

30 dispatched and identifies instruction group stops within the bundle. A bundle need not include any instruction group stops in which case the three instructions may be

executed in parallel with some or all the instructions of the next bundle.

Figure 2 is an exemplary diagram illustrating instruction slots and template maps in accordance with the present invention. The double vertical lines in the figure represent stops which may be at the end of a bundle or at an intermediate point in the bundle.

An instruction group is a sequence of instructions starting at a given bundle address and slot number and including all instructions at sequentially increasing slot numbers and bundle addresses up to the first stop, taken branch, or fault. In IA64, instructions may be of six different types:

1) A, Integer Arithmetic Logic Unit (ALU);

2) I, Non-ALU Integer;

3) M, Memory;

4) F, Floating-point;

5) B, Branch; and

6) LX, Long immediate (this is used for generating 64 bit constants and long branches although the latter is not implemented on Itanium).

IA64 execution units may be of four different types:

1) Integer (I-unit), which can execute A, I and LX instructions;

2) Memory (M-unit), which can execute M and A instructions;

3) Floating-point (F-unit), which can execute F instructions; and

4) Branch (B-unit), which can execute B instructions.

In view of the above architecture, and resource limitations of the Itanium implementation of the IA64

Docket No. AUS9-2000-0587-US1

architecture, certain combinations of instructions may be grouped for efficient parallel execution by the IA64 architecture execution units. **Table 1** shows the various instruction groups that are currently supported by the

5 Itanium implementation of the IA64 architecture. Note that the LX instruction occupies two slots.

MMF	Memory, Memory, Floating-point
MLX	Memory, Long immediate
MMI	Memory, Memory, Integer
MII	Memory, Integer, Integer
MFI	Memory, Floating-point, Integer
MMB	Memory, Memory, Branch
MFB	Memory, Floating-point, Branch
MIB	Memory, Integer, Branch
MBB	Memory, Branch, Branch
BBB	Branch, Branch, Branch

Table 1 - Currently Supported Bundles

10

As mentioned above, instruction group stops may occur after the final instruction of any of the bundles described by the preceding templates. Additionally, there are two templates that provide for stopping an

15 instruction group prior to the last instruction in the bundle. These two templates are MI_I and M_MI, where the underscore indicates the location of the instruction stop. These two templates may also identify stops at the end of the bundle.

20 In addition to the above, the Itanium implementation of the IA64 architecture has asymmetric instruction units with regard to which instructions they can execute. For example, Itanium has 2 integer units (I0, I1) where I0 can execute the entire set of A, I and LX instructions

25 but I1 is unable to execute certain specific instructions

Docket No. AUS9-2000-0587-US1

in that set such as extr and tbit. When dispatching
instructions, the first I-unit instruction in the
instruction group will be dispatched to I0 with the
second going to I1. If I1 cannot execute the instruction
5   because it is can only execute in I0, a stall occurs
until the instruction in I0 is completed at which time
the second I instruction will be dispatched to I0.

The same situation holds for the two floating point
units, i.e. F0 can execute everything, F1 can execute
10 only a subset) and the two memory units of the Itanium
implementation of IA64. To avoid these stalls, the
bundles must be arranged so that the restricted
instructions occur ahead of non-restricted instructions.
The restricted instructions are assigned type I0, F0 and
15 M0.

It is important to note that, in the following
description, NOP (pronounced "no-op") instructions may be
dispatched to designated execution units. Any unit
processing a NOP will be unavailable to process other
20 instructions in the current instruction group. A NOP
instruction is essentially an instruction that performs
no appreciable function other than to make an instruction
bundle meet architectural requirements and thereby make
an associated execution unit unavailable.

25 It is also important to note that bundles may be
created that are legal but degrade performance. For
example, in Itanium, if an instruction group includes
two M instructions, 1 I instruction and 1 A instruction,
some legal but inefficient groupings include: A) MMI
30 MII_, B) MII MII_, C) MMF MFI_, and D) MII MFI_. Again,
the underscore indicates the presence and postion of the
stop bit. These bundle pairs are inefficient because they

Docket No. AUS9-2000-0587-US1

oversubscribe the available execution units and cause
stalls. Itanium has 2 M-units, 2 I-units, 2 F-units and
3 B-units. Therefore examples A) and C) will stall when
the third M instruction is encountered while examples

5   B)and D) will stall when the third I instruction is
encountered. It makes no difference that the 3rd M and
3rd I instructions are NOPs. They still must be
dispatched to an execution unit and will cause a stall if
no unit is available.

10   Some legal and efficient bundles for the same input
include MII MFB, MFI MFI, and MIB MIB. These bundle
pairs are efficient in that there are sufficient
execution units to allow concurrent execution with no
stalls.

15   The present invention provides a mechanism to
quickly organize instructions into valid bundles that
will efficiently exploit the resources of the target
processor. With the present invention, bundle creation
is the final step in compilation and the bundles are

20   emitted, i.e. code is generated for the instructions,
directly into a code buffer associated with the
processor. The input to the bundle creation is a stream
of intermediate instructions organized into instruction
groups by a previous operation. The step of organizing

25   intermediate instructions into instruction groups may be
performed, for example, using the apparatus and method
described in co-pending and commonly assigned U.S. Patent
Application Serial No. _____ (Attorney Docket No.
AUS9-2000-0586-US1), which is hereby incorporated by

30   reference. It is assumed that the creator of the
instruction groups is aware of the limitations and
special requirements of the target implementation and

that the instruction groups will not include instruction combinations that will force oversubscription of hardware resources assuming optimal bundling is performed. The end of each instruction group is

5   identified by a stop flag set to one in the final intermediate instruction of each instruction group.

With the present invention, prior to performing the instruction bundle creation, the apparatus and method of the present invention gathers information about the

10   underlying architecture for use in the instruction bundle creation. The information gathered includes the number of each type of execution unit available and the number of bundles that can be dispatched concurrently by the architecture. For example, Itanium has two I-units, two

15   M-units, two F-units, and three B-units and can dispatch a maximum of two bundles concurrently. As described in the incorporated U.S. Patent Application Serial No. _____ (Attorney Docket No. AUS9-2000-0586-US1), this information may be obtained as part of or previous to the

20   step of instruction group creation.

Once the architecture limitation information is obtained, instruction bundle creation may be performed. The instruction bundle creation is performed an instruction group at a time. Thus, an instruction group

25   is fed to the instruction bundle creation apparatus/ method, instruction bundle creation is performed, the instruction bundle is output to a code buffer, and the next instruction group is provided to the instruction bundle creation apparatus/method. This process is

30   repeated for each instruction group in the input instruction stream.

The instruction bundle creation follows a number of

Docket No. AUS9-2000-0587-US1

rules:

1) Instructions of the same instruction type will preserve there original order. This allows instruction group creation to allow write-after-read (legal in IA64) for instructions of the same type within the same group; For example if the original programming order were:

```
mov  r5 = 1
;;
mov  r4 = r5
mov  r5 = 2
;;
```

where ;; = stop bit, when these 2 instruction groups complete, r4 will contain a 1. If the final 2 instructions were inverted it is architecturally unpredictable what would be in r4.

2) Branches will normally appear only in the final bundle of an instruction group as they can dynamically terminate the group and in implementations such as Itanium, bundles MFB, MIB and MMB will always cause a split issue (stall) after the B syllable. ;

3) For machines where the number of M-units is equal or less than the number of concurrent bundles, MM templates will only be used when there are 3 or fewer instructions remaining in the group (note that each bundle type except BBB requires an M-unit instruction);

4) Instructions are taken in order of their flexibility in terms of where that instruction can be placed in the available bundle types; and

5) for Itanium, avoid MBB and BBB templates when only a single B instruction remains because Itanium employs less efficient branch prediction for these template types.

In view of these rules, the instruction bundle
creation process will now be described. The bundle
creation process is performed based on a most restrictive
instruction type placement and proceeds to less
5   restrictive instruction type placement. The following
description of a preferred embodiment of the present
invention is provided only for illustrative purposes and
is not meant to imply any limitation on the order or type
of bundle creation checks performed.

10   The following description is provided based on the
Itanium implementation of the IA64 architecture. The
following description makes reference to a plurality of
different checks with the results being various templates
for instruction bundles. It should be appreciated that
15   once an instruction bundle template is determined in the
manner set forth below, instructions are assigned to the
bundles in accordance with the instruction bundle
templates.

As a preliminary step to the instruction bundle
20   creation, the number of instruction types in the
instruction group is counted and stored as TypesA,
TypesM, TypesI, TypesB, TypesF, TypesLX. Two additional
counters, TypesMIA and TypesALL, are incremented
concurrently with the individiual counters as
25   apporopriate. TypesAll is incremented for each
instruction type and TypesMIA is incremented when any of
TypesM, TypesI or TypesA is incremented. Once the number
of instruction types in the instruction group is known,
bundle creation is performed beginning with a check for
30   the most common instruction combinations and proceeding
to an algorithm based on a most restrictive instruction
type placement and proceeding to less restrictive

instruction type placement. As instructions are selected for inclusion in bundles the corresponding counters are decremented.

An overview of the bundling process is laid out below. The details of the process are exposed in the pseudo code figures and the descriptions thereof.

To promote efficient bundling, the most common case is handled first. This is where all the (remaining) instructions in the group are of the type M, I or A. When many instructions remain, these instructions will be packaged into several MMI and MII bundles. MMI bundles will be generated as long as the majority of remaining instructions are of type M. Otherwise, MII bundles will be generated.

For small instruction groups (or when only a few instructions remain in a large instruction group) special care is taken to insure that the bundling does not introduce hardware oversubscription. This involves examination of both the current mix of instructions and the previous instruction bundle type. It may also result in a "request" to form a partial bundle. This occurs when there are 2 or fewer instructions remaining in the group and there are fewer than two I type instructions.

In this case, to improve instruction cache footprint, it may be best to package the remaining instructions into MI_I or (when there is a single M or A) M_MI bundles. However, the final determination cannot be made until the subsequent instruction group is examined to determine if such packaging would have an adverse effect on overall performance. Instead a variable is set and passed to the next iteration of the instruction bundler to indicate that the previous group could be

Docket No. AUS9-2000-0587-US1

terminated with M_MI or MI_I. If that subsequent group
can make good use of the available bundle fragments (MI
or I) then the M_MI or MI_I bundles will be formed.
Otherwise, the preceding bundle is completed by inserting
5   appropriate NOP(s) and assigning a template that has no
intra-bundle stop but has a stop at the end of the
bundle. In this way, the new instruction group will
start with a "fresh" bundle. For example, on Itanium, if
the preceding instruction group indicates that it could
10   be concluded in an MI_I bundle and the current
instruction group includes: 2 M types, 2 I types and 1 B
type, then one of the I types could be included in the
previous bundle. However, because of the remaining mix
of instructions an additional two bundles would be
15   required to hold the remaining instructions. For example
the group could be packaged as MI_I MFB MIB_. Note that
second instruction group would span more than the 2
bundle dispersal window of Itanium and would require a
minimum of two cycles to complete. Instead, the current
20   invention would reject the invitation to bundle the
preceding group with an MI_I bundle and would cause the
instructions to be organized as MFI_ MII MFB_.
Thus the second group could be executed in a single cycle
with no adverse effects on the preceding group.
25   After the common cases are handled the focus is
shifted to the most restrictive instruction types. For
Itanium, if there are instructions that must run in I0 or
F0 they are placed in bundles first. Otherwise, if there
are any LX instructions in the instruction group, it is
30   known that only a MLX bundle can contain a LX instruction
and therefore a MLX bundle is created.
         The next most restrictive instruction type to

Docket No. AUS9-2000-0587-US1

consider is branch. In that the branches must appear as
the final instructions in the instruction group, a
determination is made as to whether there are any B
(branch) instructions and whether there are 2 or fewer
5   instrucions of all other types. If there are B
instructions and two or fewer other instructions, a
determination is made as to whether there are any F
instructions in the instruction group. If there are two
F instructions or an F and an I instruction, it is known
10   that the inclusion of the branches must be postponed as
there is no bundle that can contain such a combination of
F and I instructions and also contain a branch. Instead,
a MFI bundle is created. Otherwise, if there is a single
F instruction, it is known that MFB would be an effective
15   instruction bundle.
    If there are no F instructions but there are B
instructions, a determination is made as to whether there
are any I instructions. If there are two I instructions,
the inclusion of branches again must be postponed as
20   there is no IIB template. Instead a MII bundle is
generated. If there are not two I instructions, but
there are two M instructions and a single B instruction a
MMB bundle is generated to end the instruction group.
Otherwise, if there are two non-B instructions they must
25   include at most one I and one M. In this case the bundle
MIB will be an effective choice.
    If there is only one B instruction, and the above
checks are not met, the bundles may effectively include a
MFB type bundle. If there are two B instructions and the
30   above checks are not met, the bundles may include a MBB
type bundle. Otherwise, if none of the above applies,
then there must be more than 2 B instructons remaining

Docket No. AUS9-2000-0587-US1

and no non-B instructions. In this case the BBB template
is the best choice.

Once the instruction bundle creation is performed
based on the LX and B instruction placement, a
5   determination is made as to whether there are any F
instructions remaining. If so, a determination is made
as to whether there are only three instructions remaining
which include two M instructions. If so, the bundles can
effectively include a MMF type instruction bundle.
10   Otherwise, if there is an I or A instruction or there are
three B instructions, the MFI template would be an
effective template for one of the bundles.

Once all instructions in the instruction group are
added to bundles using the process described above, a
15   stop bit is inserted unless an M_MI or MI_I bundle is
being proposed. In either case, if more instruction
groups remain the process is repeated. Otherwise, if an
M_MI or MI_I bundle has been proposed, the unfinished
bundle is completed by inserting appropriate NOP(s) and
20   assigning template MFB_ for M_MI and MIB_ for template
MI_I. At that point the process is complete.

Figures 3A-3C are diagrams illustrating pseudo-code
for performing the method of the present invention for
Itanium. The following terms, used in **Figures 3A-3C**,
25   have the following associated meanings:

Take-M: take first remaining M0 instruction; if
none, take first remaining M instruction; if none, take
first remaining A instruction; if none, insert NOP
instruction.

30   Take-I: take first remaining I0 instruction; if
none, take first remaining I instruction; if none, take
first remaining A instruction; if none, insert NOP.

Docket No. AUS9-2000-0587-US1

Take-F: take first remaining F0 instruction; if none, take first remaining F instruction; if none, insert NOP;

Take-LX: take the first remaining LX instruction

5   (take-LX is never executed when there is no LX instruction remaining).

Take-B: take the first remaining B instruction; if none, insert NOPB (branch has its own special NOP, described below).

10   NOP: insert NOP (note execution units M, I and F share the same NOP instruction).

NOPB: insert branch NOP (this NOP has the same effect as NOP but has a different opcode and format).

The pseudo-code shown in **Figures 3A-3C** provide one

15   possible implementation of the present invention and is not meant to be limiting in any way. Essentially, the pseudo-code in **Figures 3A-3C** provide a series of checks and associated bundle templates into which instructions are bundled.

20   First the most common cases are handled and then the checks flow from a most restrictive type of instruction placement to a less restrictive type instruction placement. When fewer than 3 instructions remain in an instruction group, a determination is made as to whether

25   an intra-bundle might be formed with the final instructions. If so, a variable, INCOMPLETE, is set to indicate the type of bundle (M_MI or MI_I) being considered. However, the completion of the bundle including template selection is postponed until the

30   counters available for the next instruction group. At that time it can be determined if instructions from the new group can effectively populate the remaining slot(s)

Docket No. AUS9-2000-0587-US1

of the previous bundle. If so the bundle suggested by
INCOMPLETE is created. Otherwise, if INCOMPLETE is M_MI
the previous bundle is finished by filling in NOPs and
assigning it the MFB_ template. Whereas if it is MI_I it
5   is completed as a MIB_ template.

   Figure 4 is a flowchart outlining an exemplary
operation of the present invention. As shown in **Figure
4**, the operation starts with receiving an instruction
group as input (step **410**). The number of each
10   instruction type is determined (step **420**). A
determination is made as to whether the variable
INCOMPLETE is zero (step **430**). If INCOMPLETE is not
zero, a determination is made as to whether the current
mix of instructions could effectively populate the
15   fragment (step **440**). If so the intra-stop bundle is
completed by inserting one or two instructions from the
current group into the previous bundle (step **450**).
Otherwise, the previous bundle is completed by inserting
NOP(s) and using template MIB_ (for MI_I) or MFB_ (for
20   M_MI) (step **460**).

   If INCOMPLETE is zero in step **430**, instruction
bundle creation is performed with regard to the efficient
bundle templates (step **470**). This may be performed in
the manner outlined above. The operation then ends.
25   Thus, the present invention provides a mechanism by
which efficiently executed instruction bundles may be
created in a quick manner. With the present invention,
very little CPU time is required to bundle instructions
from instruction groups in an optimized manner such that
30   execution of the instruction bundles maximizes the
efficiency of the overall system.

Docket No. AUS9-2000-0587-US1

It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of

5   the present invention are capable of being distributed in the form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the particular type of signal bearing media actually used to carry out the

10 distribution. Examples of computer readable media include recordable-type media such a floppy disc, a hard disk drive, a RAM, and CD-ROMs and transmission-type media such as digital and analog communications links.

The description of the present invention has been

15 presented for purposes of illustration and description, but is not intended to be exhaustive or limited to the invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiment was chosen and described in

20 order to best explain the principles of the invention, the practical application, and to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use contemplated.

Docket No. AUS9-2000-0587-US1

CLAIMS:

What is claimed is:

1.   A method for creating instruction bundles,

5   comprising:

   receiving an instruction group having one or more instructions;

   determining a number of each possible type of instruction in the one or more instructions of the

10   instruction group; and

   creating one or more instruction bundles based on the number of each possible type of instruction in the one or more instructions of the instruction group.

15   2.   The method of claim 1, wherein receiving an instruction group having one or more instructions includes receiving a stream of intermediate instructions organized into instruction groups.

20   3.   The method of claim 1, further comprising gathering information about an architecture for use in creating instruction bundles.

4.   The method of claim 3, wherein the information

25   includes at least one of a number of each type of execution unit available in the architecture and a number of bundles that can be dispatched concurrently by the architecture.

30   5.   The method of claim 2, wherein the steps of determining a number of each possible type of instruction and creating one or more instruction bundles is performed

Docket No. AUS9-2000-0587-US1

for each instruction group in the stream of intermediate instructions.

6.   The method of claim 1, wherein creating one or more
instruction bundles is performed in view of one or more
of the following rules:

     1) instructions of the same instruction type will
preserve there original order;

     2) branches will normally appear only in the final
bundle of an instruction group;

     3) for architectures where a number of M execution
units is equal or less than a number of concurrent
bundles, MM templates will only be used when there are
three or fewer instructions remaining in the group;

     4) instructions are taken in order of their
flexibility in terms of where that instruction can be
placed in the available bundle types; and

     5) MBB and BBB templates are avoided when only a
single B instruction remains.

7.   The method of claim 1, wherein creating one or more
instruction bundles is performed based on a most common
instruction combination first.

8.   The method of claim 7, wherein creating one or more
instruction bundles is performed based on a most
restrictive instruction type placement and proceeds to
less restrictive instruction type placement second.

9.   The method of claim 1, wherein creating one or more
instruction bundles is performed based on a most
restrictive instruction type placement and proceeds to

Docket No. AUS9-2000-0587-US1

less restrictive instruction type placement.

10. The method of claim 1, wherein determining a number of each possible type of instruction in the one or more
5 instructions of the instruction group includes incrementing instruction counters based on the number of each possible type of instruction in the one or more instructions, and wherein creating one or more instruction bundles includes decrementing the instruction
10 counters as instructions are added to instruction bundles.

11. The method of claim 7, wherein the most common instruction combination is where all instructions in the
15 instruction group are of a memory instruction type, integer instruction type or integer arithmetic logic unit type.

12. The method of claim 1, wherein creating one or more
20 instruction bundles includes ensuring that creating the one or more instruction bundles does not introduce hardware oversubscription.

13. The method of claim 12, wherein ensuring that
25 creating the one or more instruction bundles does not introduce hardware oversubscription includes forming partial instruction bundles.

14. An apparatus for creating instruction bundles,
30 comprising:
        means for receiving an instruction group having one or more instructions;

means for determining a number of each possible type of instruction in the one or more instructions of the instruction group; and

means for creating one or more instruction bundles
5   based on the number of each possible type of instruction in the one or more instructions of the instruction group.

15. The apparatus of claim 14, wherein the means for receiving an instruction group having one or more
10   instructions includes means for receiving a stream of intermediate instructions organized into instruction groups.

16. The apparatus of claim 14, further comprising means
15   for gathering information about an architecture for use in creating instruction bundles.

17. The apparatus of claim 16, wherein the information includes at least one of a number of each type of
20   execution unit available in the architecture and a number of bundles that can be dispatched concurrently by the architecture.

18. The apparatus of claim 15, wherein the means for
25   determining a number of each possible type of instruction and means for creating one or more instruction bundles operate on each instruction group in the stream of intermediate instructions.

30   19. The apparatus of claim 14, wherein the means for creating one or more instruction bundles operates in view of one or more of the following rules:

Docket No. AUS9-2000-0587-US1

1) instructions of the same instruction type will preserve there original order;

2) branches will normally appear only in the final bundle of an instruction group;

5   3) for architectures where a number of M execution units is equal or less than a number of concurrent bundles, MM templates will only be used when there are three or fewer instructions remaining in the group;

4) instructions are taken in order of their
10   flexibility in terms of where that instruction can be placed in the available bundle types; and

5) MBB and BBB templates are avoided when only a single B instruction remains.

15   20.   The apparatus of claim 14, wherein the means for creating one or more instruction bundles creates the one or more instruction bundles based on a most common instruction combination first.

20   21.   The apparatus of claim 20, wherein the means for creating one or more instruction bundles creates the one or more instruction bundles based on a most restrictive instruction type placement and proceeds to less restrictive instruction type placement second.

25

22.   The apparatus of claim 14, wherein the means for creating one or more instruction bundles creates the one or more instruction bundles based on a most restrictive instruction type placement and proceeds to less
30   restrictive instruction type placement.

23.   The apparatus of claim 14, wherein the means for

Docket No. AUS9-2000-0587-US1

determining a number of each possible type of instruction
in the one or more instructions of the instruction group
includes means for incrementing instruction counters
based on the number of each possible type of instruction

5   in the one or more instructions, and wherein the means
for creating one or more instruction bundles includes
means for decrementing the instruction counters as
instructions are added to instruction bundles.

10   24.   The apparatus of claim 20, wherein the most common
instruction combination is where all instructions in the
instruction group are of a memory instruction type,
integer instruction type or integer arithmetic logic unit
type.

15

25.   The apparatus of claim 14, wherein the means for
creating one or more instruction bundles includes means
for ensuring that creating the one or more instruction
bundles does not introduce hardware oversubscription.

20

26.   The apparatus of claim 25, wherein the means for
ensuring that creating the one or more instruction
bundles does not introduce hardware oversubscription
includes means for forming partial instruction bundles.

25

27.   A computer program product in a computer readable
medium for creating instruction bundles, comprising:
     first instructions for receiving an instruction
group having one or more instructions;

30   second instructions for determining a number of each
possible type of instruction in the one or more
instructions of the instruction group; and

Docket No. AUS9-2000-0587-US1

third instructions for creating one or more
instruction bundles based on the number of each possible
type of instruction in the one or more instructions of
the instruction group.

5

28.   The computer program product of claim 27, wherein
the first instructions for receiving an instruction group
having one or more instructions includes instructions for
receiving a stream of intermediate instructions organized
10   into instruction groups.

29.   The computer program product of claim 27, further
comprising fourth instructions for gathering information
about an architecture for use in creating instruction
15   bundles.

30.   The computer program product of claim 29, wherein
the information includes at least one of a number of each
type of execution unit available in the architecture and
20   a number of bundles that can be dispatched concurrently
by the architecture.

31.   The computer program product of claim 28, wherein
the second instructions for determining a number of each
25   possible type of instruction and third instructions for
creating one or more instruction bundles are executed on
each instruction group in the stream of intermediate
instructions.

30   32.   The computer program product of claim 27, wherein
the third instructions for creating one or more
instruction bundles are executed in view of one or more

Docket No. AUS9-2000-0587-US1

of the following rules:

 1) instructions of the same instruction type will preserve there original order;

 2) branches will normally appear only in the final

5 bundle of an instruction group;

 3) for architectures where a number of M execution units is equal or less than a number of concurrent bundles, MM templates will only be used when there are three or fewer instructions remaining in the group;

10 4) instructions are taken in order of their flexibility in terms of where that instruction can be placed in the available bundle types; and

 5) MBB and BBB templates are avoided when only a single B instruction remains.

15

33. The computer program product of claim 27, wherein the third instructions for creating one or more instruction bundles creates the one or more instruction bundles based on a most common instruction combination

20 first.

34. The computer program product of claim 33, wherein the third instructions for creating one or more instruction bundles creates the one or more instruction

25 bundles based on a most restrictive instruction type placement and proceeds to less restrictive instruction type placement second.

35. The computer program product of claim 27, wherein

30 the third instructions for creating one or more instruction bundles creates the one or more instruction bundles based on a most restrictive instruction type

placement and proceeds to less restrictive instruction
type placement.

36.   The computer program product of claim 27, wherein
the second instructions for determining a number of each
possible type of instruction in the one or more
instructions of the instruction group includes
instructions for incrementing instruction counters based
on the number of each possible type of instruction in the
one or more instructions, and wherein the third
instructions for creating one or more instruction bundles
includes instructions for decrementing the instruction
counters as instructions are added to instruction
bundles.

37.   The computer program product of claim 33, wherein
the most common instruction combination is where all
instructions in the instruction group are of a memory
instruction type, integer instruction type or integer
arithmetic logic unit type.

38.   The computer program product of claim 27, wherein
the third instructions for creating one or more
instruction bundles includes instructions for ensuring
that creating the one or more instruction bundles does
not introduce hardware oversubscription.

39.   The computer program product of claim 38, wherein
the instructions for ensuring that creating the one or
more instruction bundles does not introduce hardware
oversubscription includes instructions for forming
partial instruction bundles.

Docket No. AUS9-2000-0587-US1

ABSTRACT OF THE DISCLOSURE

APPARATUS AND METHOD FOR CREATING INSTRUCTION GROUPS FOR EXPLICITLY PARALLEL ARCHITECTURES

5

An apparatus and method for creating instruction groups for explicitly parallel architectures is provided. The apparatus and method accept instruction groups as input and determine a number of each possible type of

10   instruction in the instruction group. Based on the number of each possible type of instruction in the instruction group, instruction bundling is performed such that the instructions in the instruction group are bundled into efficiently executed bundles. The

15   instruction bundling further accommodates intra-bundle stop bundles in the event that more efficient bundles are not possible. The instruction bundling is performed based on a most restrictive instruction type placement first and proceeds to less restrictive instruction type

20   placement.
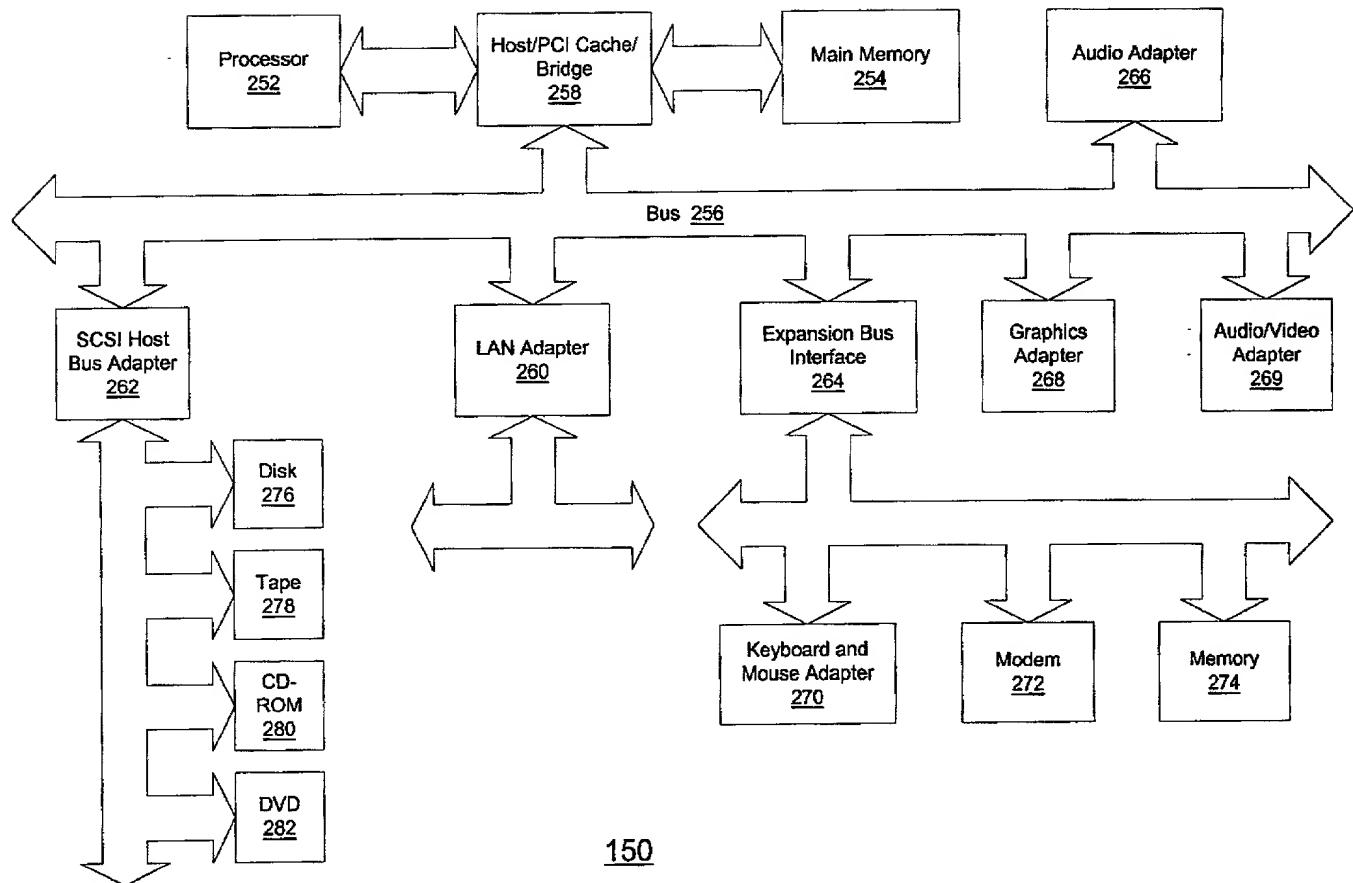
150

Figure 1

AUS9-2000-0587-US1
Sheet 1 of 6

Template	Slot 0	Slot 1	Slot 2
00	M-unit	I-unit	I-unit
01	M-unit	I-unit	I-unit
02	M-unit	I-unit	I-unit
03	M-unit	I-unit	I-unit
04	M-unit	L-unit	X-unit
05	M-unit	L-unit	X-unit
06			
07			
08	M-unit	M-unit	I-unit
09	M-unit	M-unit	I-unit
0A	M-unit	M-unit	I-unit
0B	M-unit	M-unit	I-unit
0C	M-unit	F-unit	I-unit
0D	M-unit	F-unit	I-unit
0E	M-unit	M-unit	F-unit
0F	M-unit	M-unit	F-unit
10	M-unit	I-unit	B-unit
11	M-unit	I-unit	B-unit
12	M-unit	B-unit	B-unit
13	M-unit	B-unit	B-unit
14			
15			
16	B-unit	B-unit	B-unit
17	B-unit	B-unit	B-unit
18	M-unit	M-unit	B-unit
19	M-unit	M-unit	B-unit
1A			
1B			
1C	M-unit	F-unit	B-unit
1D	M-unit	F-unit	B-unit
1E			
1F			

Figure 2

```
MainLoop:

While there are unprocessed instruction groups {
    Select next instruction group.

TopOfGroup:
    For each instruction in the group:

        switch on instruction_type {
          case TypeA:
              TypesA++;
              TypesMIA++;
            break;
          case TypeM:
              TypesM++;
              TypesMIA++;
            break;
          case TypeI:
              TypesI++;
              TypesMIA++;
            break;
          case TypeB:
              TypesB++;
            break;
          case TypeF:
              TypesF++;
            break;
          case TypeLI:
              TypesLI++;
            break;
          default:
            error;
        }
        TypesALL++;

        /*----------------------------------------------+/
        | Test for the a previous incomplete bundle
        +----------------------------------------------*/
        if INCOMPLETE is not zero {
         if INCOMPLETE equals M_MI{
            INCOMPLETE = 0;

            if despersal window is large {
MakeM_MI:
                Template = M_MI;
                take-M;
                take-I;
                goto StoreBundle;
            }
            remainder = size of instruction group % 3;

        if remainder = 0 then: {
            if TypesI > 0 AND TypesF < TypesF-units AND TypesM+TypesA <
bundle count then:
            goto MakeM_MI;
            else
             goto MakeMFB;
```

```
        }

        if remainder = 1 then: {
            if (TypesI > 0 OR TypesA > 0)  AND TypesF < TypesF-units AND
TypesM+TypesA < bundle count then:
              goto MakeM_MI;
            else
              goto MakeMFB;
        }
        /* remainder = 2 */
        if (TypesI > 0 OR TypesA > 0)  AND TypesF < TypesF-units AND
TypesM+TypesA >= bundle count then:
              goto MakeM_MI;

MakeMFB:
        Template = MFB_;
        nop;
        nopb;
        goto StoreBundle;
      /*-----------------------------------------------------------------
---+/
      | INCOMPLETE equals MI_I
      +-----------------------------------------------------------------
---*/
      } else {
        INCOMPLETE = 0;
        if despersal window is large {
MakeMI_I:
            Template = MI_I;
            take-I;
            goto StoreBundle;
        }
        remainder = size of instruction group % 3
        if remainder = 2 then:
            goto MakeMIB;

        if remainder = 0 then: {
            if TypesI > 0 AND TypesF < TypesF-units
            AND TypesM+TypesA < bundle count then: goto MakeMI_I;
            else   goto MakeMIB;
        }
        /* remainder = 1 */
        if (TypesI > 0 OR TypesA > 0)  AND TypesF < TypesF-units
        AND TypesM+TypesA < bundle count then: goto MakeMI_I;

MakeMIB:
            Template = MIB_;
            nopB;
            goto StoreBundle;
      }

    }


 While TypesALL > 0{ // while instructions remain in group
```

Figure 3B

```
        if TypesALL is equal to TypesMIA {
         if TypesALL > 3 {
              if TypesM > TypesI+TypesA then: Template = MMI; take-M;take-M;take-I
              else Template = MII; take-M;take-I;take-I
              goto StoreBundle;
         }
         if TypesALL = 3 and this is the first bundle of the group {
              if TypesI > 1 then: Template = MII; take-M;take-I;take-I
              else Template = MMI; take-M;take-M;take-I
              goto StoreBundle;
         }
         if TypesALL = 2  OR TypesALL = 1 and TypesI = 1{
              if TypesM = 2 then: Template = MMF; take-M;take-M;nop; goto
StoreBundle;
              else INCOMPLETE = MI_I take-M; take-I; goto TopOfGroup
         }
         /*  TypesALL = 1 */
              INCOMPLETE = M_MI take-M; goto TopOfGroup
        }
        if TypesLX > 0 then:  Template= MLX;  take-M;  take-LX; goto StoreBundle;

        if TypesB > 0 AND TypesALL-TypesB < 3 then: {
         if typesF > 0 then: {
              if TypesF+TypesI = 2 then: Template= MFI  nop;takeF;take-I; goto
StoreBundle;
              · else Template=MFB take-M;takeF;take-B; goto StoreBundle;
         } else if TypesI > 0 {
              if TypesI = 2 then: Template= MII  nop;takeF;take-I; goto StoreBundle;
              else Template=MIB take-M;take-I;take-B; goto StoreBundle;
         }else if TypesM = 2 then:  Template = MMB take-M;take-M;take-B; goto
StoreBundle;
         else if TypesALL-TypesB = 2 then: Template = MIB take-M;take-I;take-B;
goto StoreBundle;
         else if TypesB = 1 then: Template = MFB take-M;takeF;take-B; goto
StoreBundle;
         else if there are 2 TypesB instructions  or 1 non-TypesB: Template = MBB
take-M;take-B;take-B; goto StoreBundle;
         else Template = BBB take-B;take-B;take-B; goto StoreBundle;
         }
         /*  TypesF > 0 */
          if TypesALL = 3 AND TypesM = 2 then: Template = MMF take-M;take-M takeF;
goto StoreBundle;
              else Template = MFI take-M;takeF take-I; goto StoreBundle;
}
storeBundle:
   if TypesALL = 0 then: insert stop bit in Template;
   build bundle in code buffer;
   if TypesALL = 0 then: goto MainLoop;
   else goto TopOGroup;
}
DONE
   if INCOMPLETE is not zero then: {
      if INCOMPLETE = M_MI then: Template = MFB; nop; nopB; goto StoreBundle;
      else Template = MIB; nopB; goto StoreBundle;
   }
```

Figure 3C

AUS9-2000-0587-US1

Sheet 5 of 6

Figure 4

```
                    ┌─────────┐
                    │  Enter  │
                    └────┬────┘
                         │
                         ▼
                ┌──────────────────┐
                │    Receive       │ ─── 410
                │ Instruction Group│
                └────────┬─────────┘
                         │
                         ▼
                ┌──────────────────┐
                │ Determine Number │ ─── 420
                │ of Each Type of  │
                │   Instruction    │
                └────────┬─────────┘
                         │
                         ▼
                ┌──────────────────┐
                │     Perform      │ ─── 430
                │   Instruction    │
                │    Bundling      │
                └────────┬─────────┘
                         │
                         ▼
                ┌──────────────────┐
                │  Attempt Intra-  │ ─── 440
                │   Bundle Stop    │
                │    Templates     │
                └────────┬─────────┘
                         │
                         ▼
                      ╱──────╲
                    ╱  Intra-  ╲
            NO    ╱    Bundle    ╲ ─── 450
           ┌─────◁    Stop        ▷
           │      ╲  Templates   ╱
           │        ╲  Apply?  ╱
           │          ╲──────╱
           │             │ YES
           │             ▼
           │     ┌──────────────────┐
           │     │ Use Stall Possible│ ─── 460
           │     │    Templates     │
           │     └────────┬─────────┘
           │              │
           │              ▼
           │           ╱──────╲
           │   YES   ╱   More   ╲ ─── 470
           └────────◁ Instructions?▷
                     ╲          ╱
                       ╲──────╱
                          │ NO
                          ▼
                    ┌─────────┐
                    │  Exit   │
                    └─────────┘
```
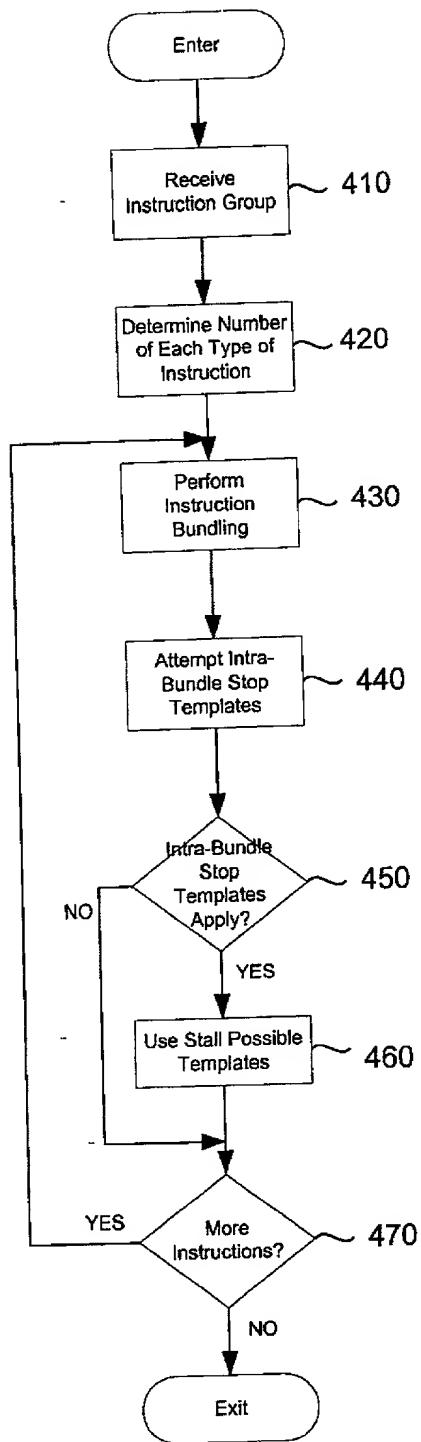
DECLARATION AND POWER OF ATTORNEY FOR

PATENT APPLICATION

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled

APPARATUS AND METHOD FOR CREATING INSTRUCTION BUNDLES IN AN EXPLICITLY PARALLEL ARCHITECTURE

the specification of which (check one)

X_ is attached hereto.

__ was filed on _____
    as Application Serial No._____
    and was amended on _____
                        (if applicable)

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the patentability of this application in accordance with Title 37, Code of Federal Regulations, §1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, §119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s):   Priority Claimed

                                                        ____ Yes____ No
_____   _____   _____
   (Number)   (Country)   (Day/Month/Year)

I hereby claim the benefit under Title 35, United States Code, §120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, §112, I acknowledge the duty to disclose information material to the patentability of this application as defined in Title 37, Code of Federal Regulations, §1.56 which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

_____   _____   _____
(Application Serial #)   (Filing Date)   (Status)

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

POWER OF ATTORNEY: As a named inventor, I hereby appoint the following attorneys and/or agents to prosecute this application and transact all business in the Patent and Trademark Office connected therewith.

John W. Henderson, Jr., Reg. No. 26,907; Thomas E. Tyson, Reg. No. 28,543; James H. Barksdale, Jr., Reg. No. 24,091; Casimer K. Salys, Reg. No. 28,900; Robert M. Carwell, Reg. No. 28,499; Douglas H. Lefeve, Reg. No. 26,193; Jeffrey S. LaBaw, Reg. No. 31,633; David A. Mims, Jr., Reg. No. 32,708; Volel Emile, Reg. No. 39,969; Anthony V. England, Reg. No. 35,129; Leslie A. Van Leeuwen, Reg. No. 42,196; Christopher A. Hughes, Reg. No. 26,914; Edward A. Pennington, Reg. No. 32,588; John E. Hoel, Reg. No. 26,279; Joseph C. Redmond, Jr., Reg. No. 18,753; Marilyn S. Dawkins, Reg. No. 31,140; Mark E. McBurney, Reg. No. 33,114; Duke W. Yee, Reg. No. 34,285; Colin P. Cahoon, Reg. No. 38,836; Stephen R. Loe, Reg. No. 43,757; Stephen J. Walder, Jr., Reg. No. 41,534; Charles D. Stepps, Jr., Reg. No. 45,880; Stephen R. Tkacs, Reg. No. 46,430, and Christopher P. O'Hagan, Reg. No. P-46,966.

Send correspondence to: Duke W. Yee, Carstens, Yee & Cahoon, LLP, P.O. Box 802334, Dallas, Texas 75380 and direct all telephone calls to Duke W. Yee, (972) 367-2001

FULL NAME OF SOLE OR FIRST INVENTOR: Geoffrey Owen Blandy

INVENTORS SIGNATURE:_____ DATE:_____9/26/2000_____
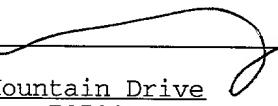
RESIDENCE: 9412 Bell Mountain Drive
             Austin, Texas 78730

CITIZENSHIP: United States

POST OFFICE ADDRESS: SAME AS ABOVE